1. Write a C program to implement the back end of the compiler

#include <stdio.h>

#include <string.h>

// Structure to store tokens (simulating a compiler backend)

typedef struct {

char lexeme[100];

char type[20];

} Token;

void generateIR(Token tokens[], int count) {

printf("\nIntermediate Representation:\n");

for (int i = 0; i < count; i++) {

printf("%s -> %s\n", tokens[i].lexeme, tokens[i].type);

}

}

int main() {

Token tokens[] = {{"int", "Keyword"}, {"main", "Identifier"}, {"(", "Symbol"}, {")", "Symbol"},

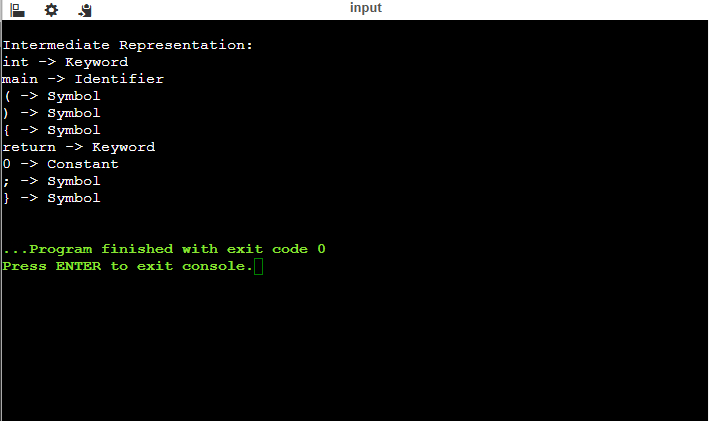
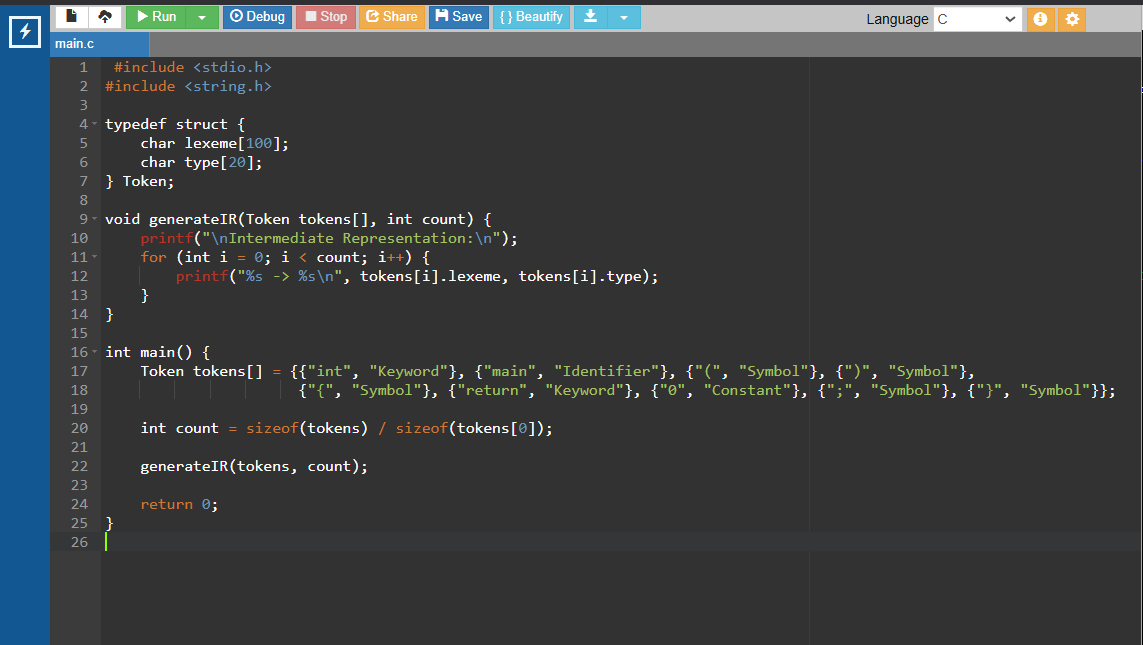
{"{", "Symbol"}, {"return", "Keyword"}, {"0", "Constant"}, {";", "Symbol"}, {"}", "Symbol"}};

int count = sizeof(tokens) / sizeof(tokens[0]);

generateIR(tokens, count);

return 0;

}



1. The lexical analyzer should ignore redundant spaces, tabs and new lines. It should also ignore comments. Although the syntax specification states that identifiers can be arbitrarily long, you may restrict the length to some reasonable value. Write a LEX specification file to take input C program from a .c file and count t he number of characters, number of lines & number of words.

**Input Source Program: (sample.c)**

#include <stdio.h>

int main()

{

int number1, number2, sum;

printf("Enter two integers: ");

scanf("%d %d", &number1, &number2);

sum = number1 + number2;

printf("%d + %d = %d", number1, number2, sum);

return 0;

}

**Code**

#include <stdio.h>

#include <ctype.h>

int main() {

FILE \*file;

char filename[100], ch;

int characters = 0, words = 0, lines = 0;

int in\_word = 0;

printf("Enter the filename: ");

scanf("%s", filename);

file = fopen(filename, "r");

if (file == NULL) {

printf("Error opening file!\n");

return 1;

}

while ((ch = fgetc(file)) != EOF) {

characters++;

if (ch == '\n') lines++;

if (isspace(ch)) {

if (in\_word) {

words++;

in\_word = 0;

}

} else {

in\_word = 1;

}

}

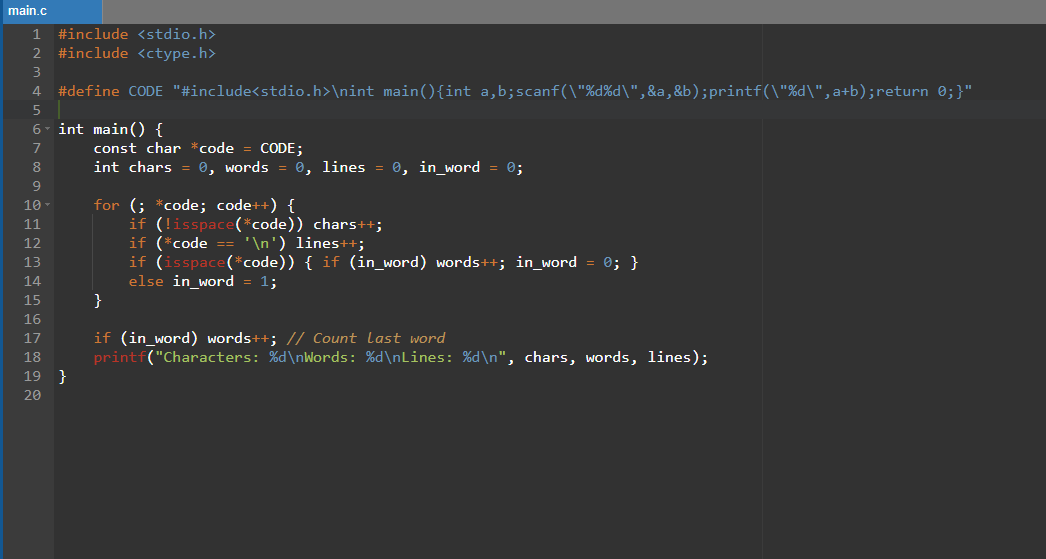
if (in\_word) words++; // Count last word if file doesn't end in whitespace

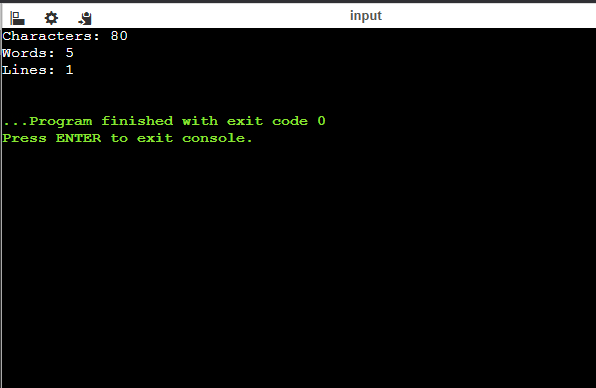
fclose(file);

printf("Characters: %d\nWords: %d\nLines: %d\n", characters, words, lines);

return 0;

}





1. Write a LEX program to print all the constants in the given C source program file.

**Input Source Program: (sample.c)**

#define PI 3.14

#include<stdio.h> #include<conio.h>

void main()

{

int a,b,c = 30;

printf("hello");

}

**Code**

#include <stdio.h>

#include <ctype.h>

int main() {

FILE \*file;

char filename[100], ch;

printf("Enter the filename: ");

scanf("%s", filename);

file = fopen(filename, "r");

if (file == NULL) {

printf("Error opening file!\n");

return 1;

}

printf("Constants found:\n");

while ((ch = fgetc(file)) != EOF) {

if (isdigit(ch)) {

printf("%c", ch);

while (isdigit(ch = fgetc(file)) || ch == '.') {

printf("%c", ch);

}

printf("\n");

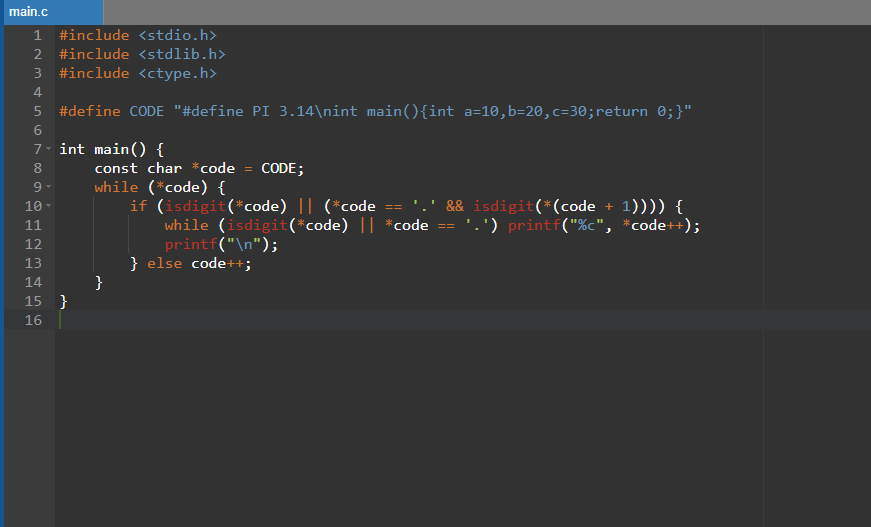
}

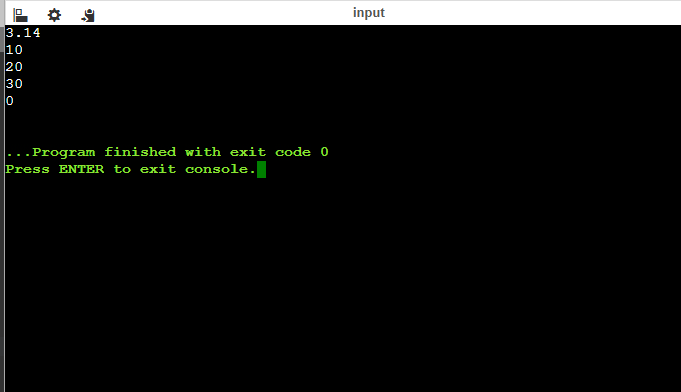
}

fclose(file);

return 0;

}





1. Write a LEX program to count the number of Macros defined and header files included in the C program.

**Input Source Program: (sample.c)**

#define PI 3.14

#include<stdio.h>

#include<conio.h>

void main()

{

int a,b,c = 30;

printf("hello");

}

**Code**

#include <stdio.h>

#include <string.h>

int main() {

FILE \*file;

char filename[100], line[256];

int macro\_count = 0, header\_count = 0;

printf("Enter the filename: ");

scanf("%s", filename);

file = fopen(filename, "r");

if (file == NULL) {

printf("Error opening file!\n");

return 1;

}

while (fgets(line, sizeof(line), file)) {

if (strstr(line, "#define")) macro\_count++;

if (strstr(line, "#include")) header\_count++;

}

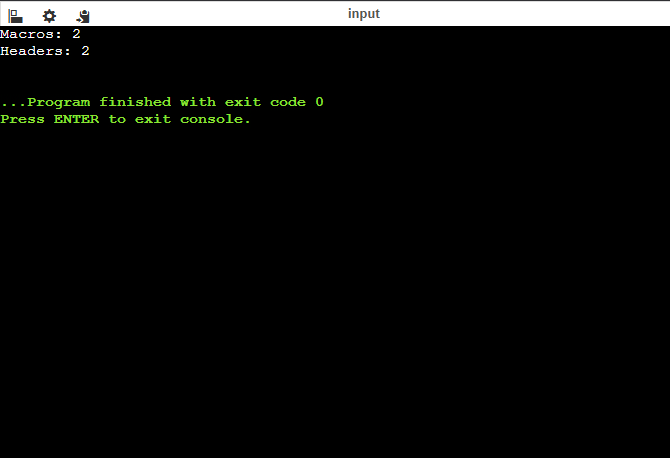
fclose(file);

printf("Macros: %d\nHeader Files: %d\n", macro\_count, header\_count);

return 0;

}





1. Write a LEX program to print all HTML tags in the input file.

**Input Source Program: (sample.html)**

<html>

<body>

<h1>My First Heading</h1>

<p>My first paragraph.</p>

</body>

</html>

**Code**

#include <stdio.h>

int main() {

FILE \*file;

char filename[100], ch;

int inside\_tag = 0;

printf("Enter the filename: ");

scanf("%s", filename);

file = fopen(filename, "r");

if (file == NULL) {

printf("Error opening file!\n");

return 1;

}

printf("HTML Tags Found:\n");

while ((ch = fgetc(file)) != EOF) {

if (ch == '<') {

inside\_tag = 1;

printf("<");

} else if (ch == '>') {

inside\_tag = 0;

printf(">\n");

} else if (inside\_tag) {

printf("%c", ch);

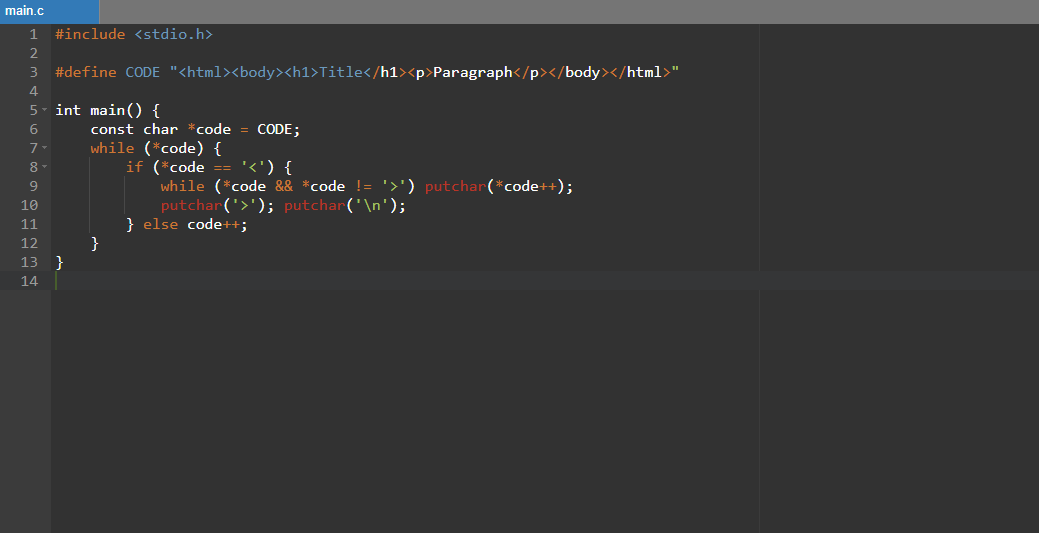
}

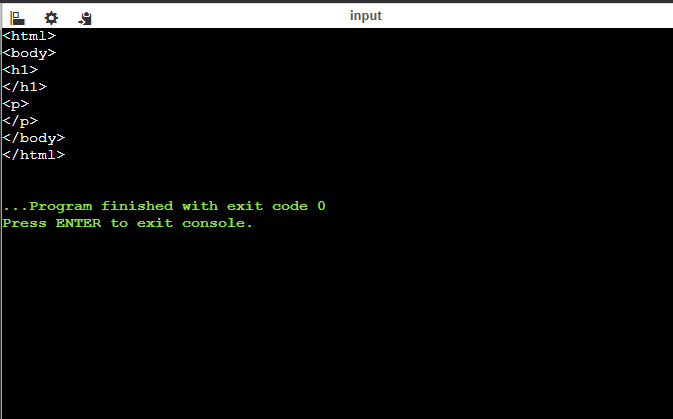
}

fclose(file);

return 0;

}





1. Write a LEX program which adds line numbers to the given C program file and display the same in the standard output.

**Input Source Program: (sample.c)**

#define PI 3.14

#include<stdio.h>

#include<conio.h>

void main()

{

int a,b,c = 30;

printf("hello");

}

**Code**

#include <stdio.h>

int main() {

FILE \*file;

char filename[100], line[256];

int line\_number = 1;

printf("Enter the filename: ");

scanf("%s", filename);

file = fopen(filename, "r");

if (file == NULL) {

printf("Error opening file!\n");

return 1;

}

while (fgets(line, sizeof(line), file)) {

printf("%d %s", line\_number++, line);

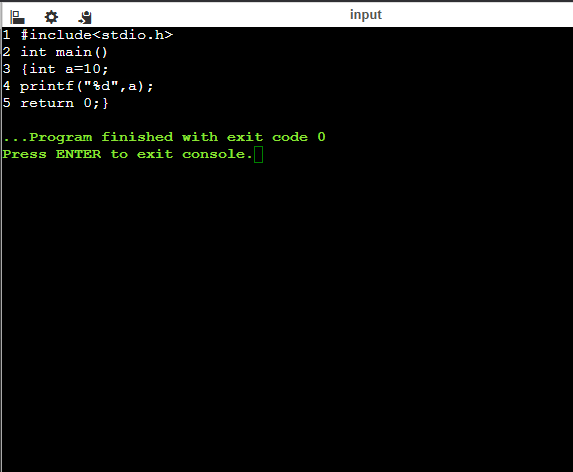
}

fclose(file);

return 0;

}





1. Write a LEX specification count the number of characters, number of lines & number of words.

**Code**

#include <stdio.h>

#include <ctype.h>

int main() {

FILE \*file;

char filename[100], ch;

int characters = 0, words = 0, lines = 0;

int in\_word = 0;

printf("Enter the filename: ");

scanf("%s", filename);

file = fopen(filename, "r");

if (file == NULL) {

printf("Error opening file!\n");

return 1;

}

while ((ch = fgetc(file)) != EOF) {

characters++;

if (ch == '\n') lines++;

if (isspace(ch)) {

if (in\_word) {

words++;

in\_word = 0;

}

} else {

in\_word = 1;

}

}

if (in\_word) words++; // Count last word if file doesn't end in whitespace

fclose(file);

printf("Characters: %d\nWords: %d\nLines: %d\n", characters, words, lines);

return 0;

}

